Texture Mapping and Blending / Compositing

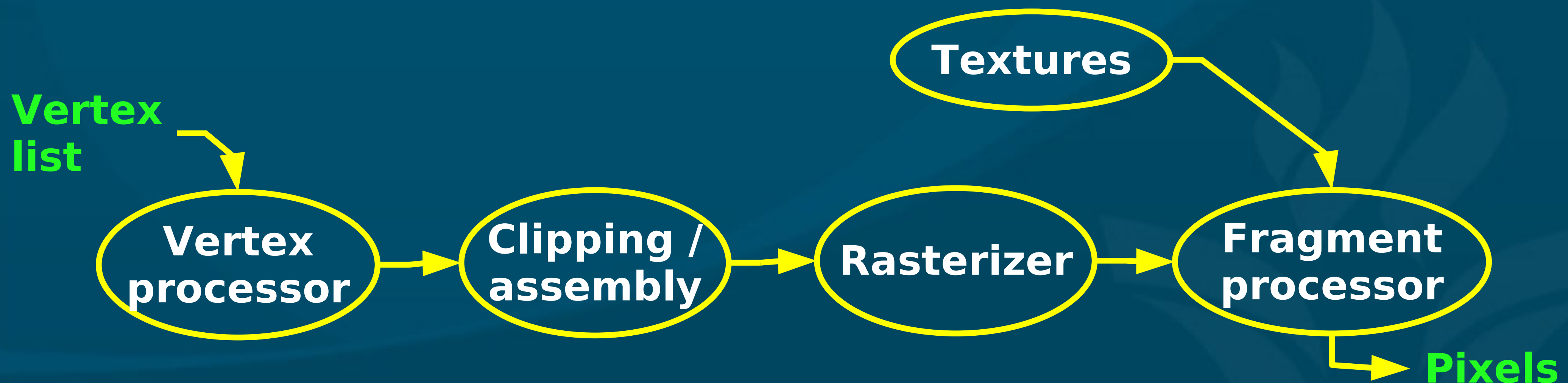
See: Edward Angel's text,
“Interactive Computer Graphics”

12 March 2009
CMPT370
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Textures in the rendering pipeline

- Texture processing happens late in the pipeline
- Relatively few polygons make it past clipping
- Each pixel on the fragment maps back to texture coordinates: (s,t) location on the texture map image



(1) Steps to create a texture

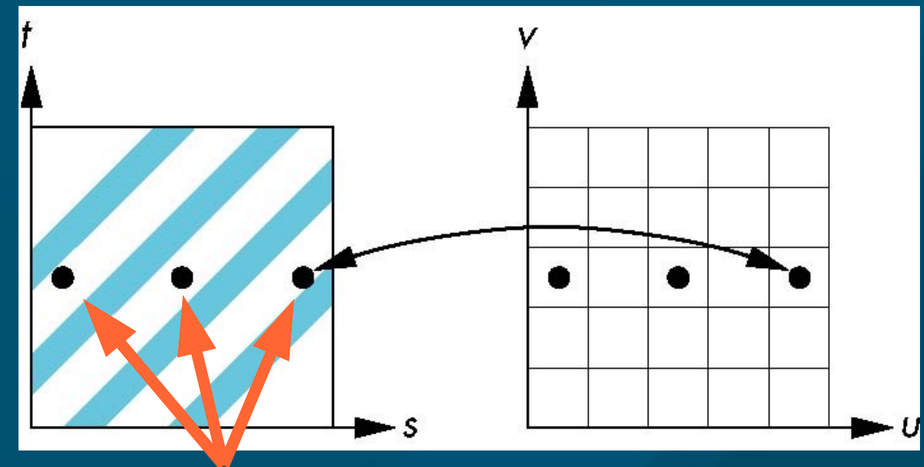
- On initialization:
  - Read or generate image to pixel array
  - Request a texture object: glGenTextures()
  - Select texture object: glBindTexture()
  - Set options (wrap, filter): glTexParameter()
  - Load image to texture: glTexImage2D()
    - Or copy from framebuffer
    - Or load with mip-maps: gluBuild2DMipmaps()

(2) Steps to use a texture

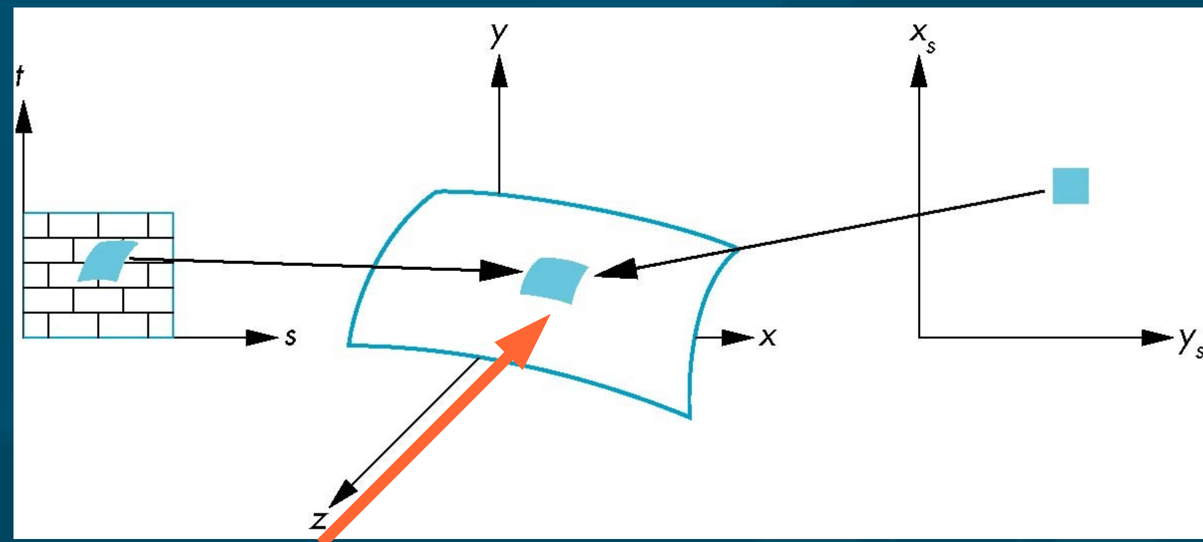
- On every frame (draw()):
  - Enable texturing: glEnable(GL_TEXTURE_2D)
  - Select texture object: glBindTexture()
  - Set blending modes: glTexEnvf()
  - Assign texture coordinates to vertices
    - glTexCoord() with each vertex
    - Or use generated texcoords: glTexGen()

Filtering: avoiding aliasing

- Pixels in fragment may map back to widely-spaced locations in texture coordinates
- Results in aliasing artifact
- Solution: average over an area: preimage of pixel
- GL provides 2x2 area averaging



misses blue bars entirely!



***preimage* of pixel in general is curved!**

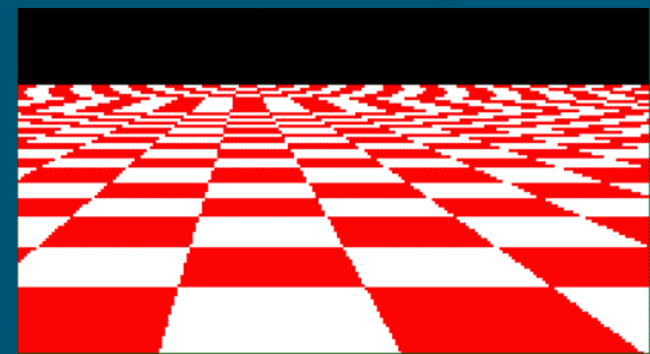
Parameters for texmaps

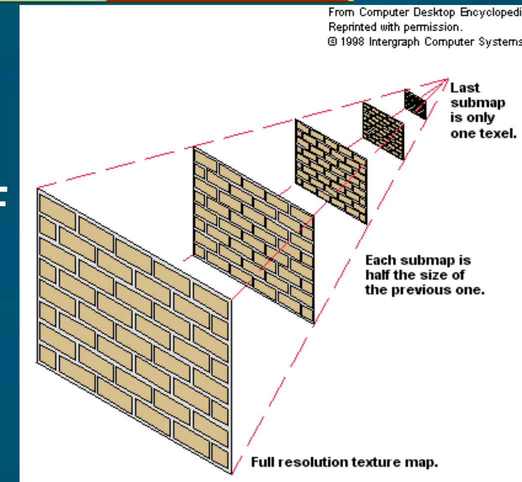
- Filtering:
  - Magnification (MAG) and minimization (MIN)
    - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  - Nearest-neighbor or GL_LINEAR interpolation
    - 2x2 interp needs 1-pixel border around texture
- Wrapping: what about texcoords outside (0,1)?
  - Repeat (tile) or clamp (border pixels)
  - In either s or t directions in texture
    - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);

Texture blending functions

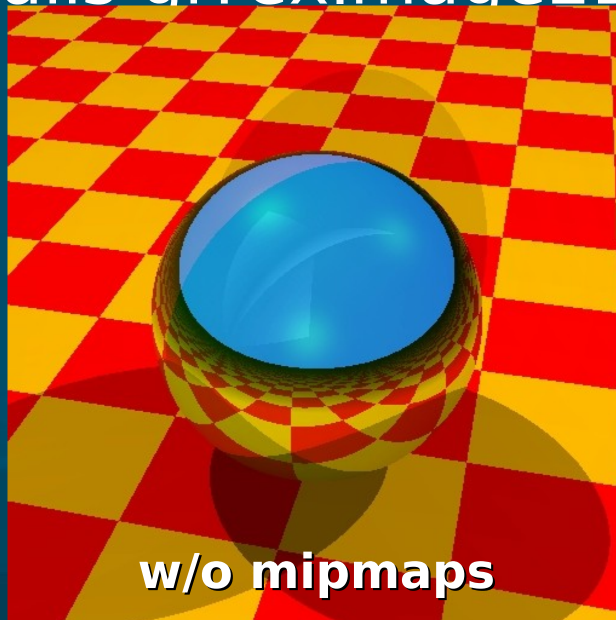
- glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
- Last param is the blending mode:
  - How the texture colour is combined with the existing colour (e.g., shaded via Gouraud)
  - GL_REPLACE: uses only this texture
  - GL_DECAL: pastes texture on top
  - GL_MODULATE: multiplies colours
  - GL_BLEND: uses texture to determine amount of blend between old colour and a fixed blend colour (set w/GL_TEXTURE_ENV_COLOR)

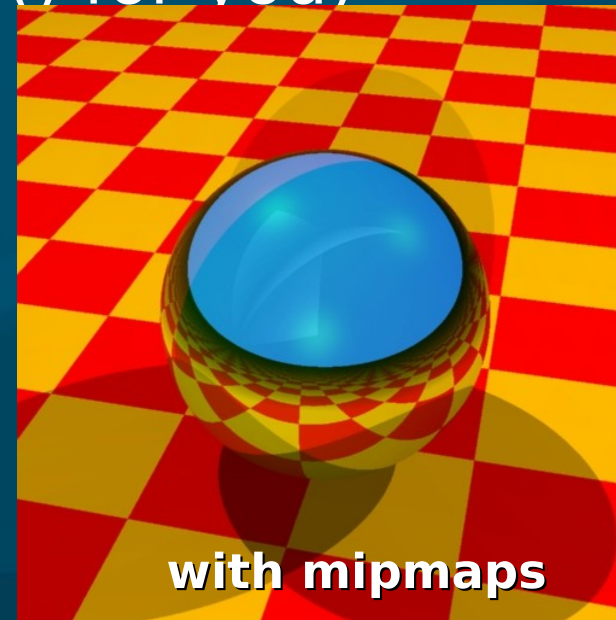
Mip-maps





- Aliasing (jaggies) can occur when textures become very small on-screen
- Pre-calculate filtered low-res versions of the texture: levels of detail (LoD)
  - Use gluBuild2DMipmaps()
    (it calls glTexImage2D() for you)



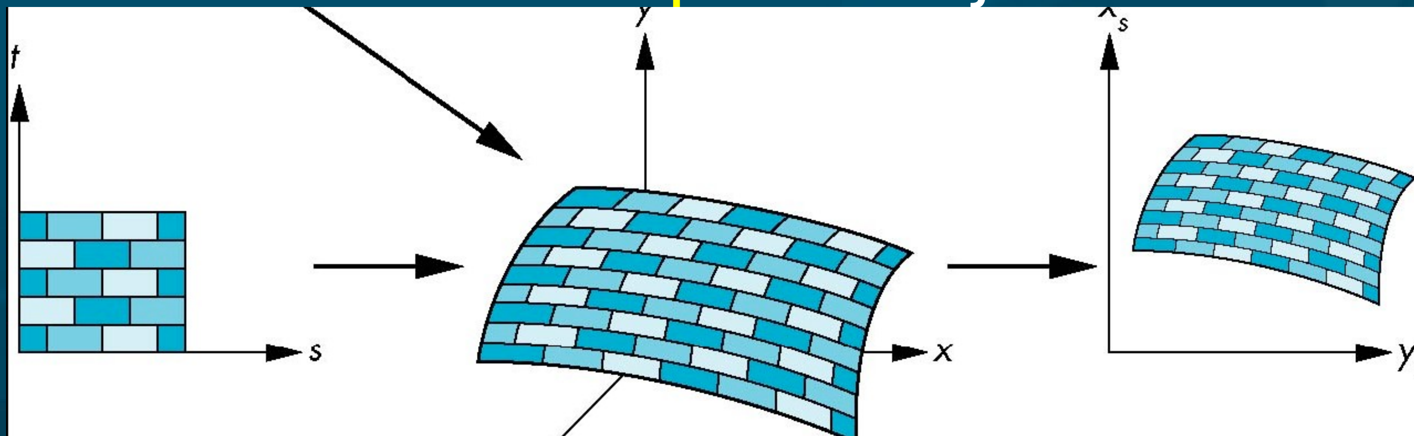


Using framebuffer as a texture

- Instead of loading a JPEG file for a texture, you can use the framebuffer itself:
    - glCopyTexImage2D(GL_TEXTURE_2D, *level*, *intFmt*, *x*, *y*, *w*, *h*, *border*)
  - Copies a rectangle from the framebuffer, starting at (x,y) with size (w,h)
  - level, intFmt, border just as in glTexImage2D
- Can use to do cheap reflections:
  - Flip model-view matrix and render
  - Texture-map framebuffer onto object

Texture coordinates

- The rectangular texture is parameterized by (s,t) in the range (0,1)
- Specify texture coordinates with each vertex
  - glTexCoord2f(0.5, 0.7)
  - Part of the OpenGL state for the vertex, just like colour / material properties
  - Texcoords are interpolated just like shades



Automatic texcoord generation

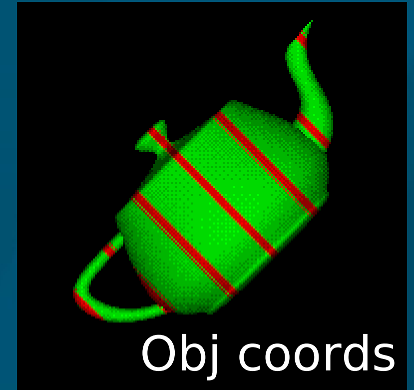
- glEnable(GL_TEXTURE_GEN_S);   // or GL_T
- glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, mode)
  - mode: GL_OBJECT_LINEAR, GL_EYE_LINEAR, or GL_SPHERE_MAP

- If mode is Object-linear: texture is fixed to object
  - Generated texcoord (s) is distance from vertex {x, y, z, w} to a reference plane $\{p_1, p_2, p_3, p_4\}$:

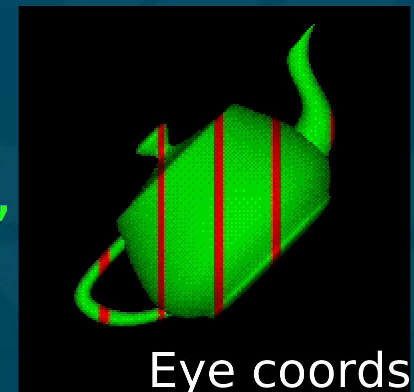
    $s = p_1 x + p_2 y + p_3 z + p_4 w$
- Teapot example (Redbook texgen.c): reference plane = {1., 1., 1., 0.} (slanted)

Object vs. eye coordinates

- If mode is GL_OBJECT_LINEAR, generated texcoords are in the model coordinate system
  - Texture fixed to object

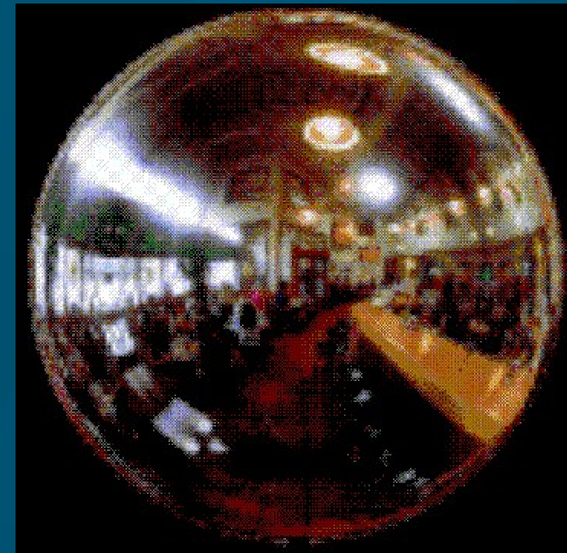

Obj coords

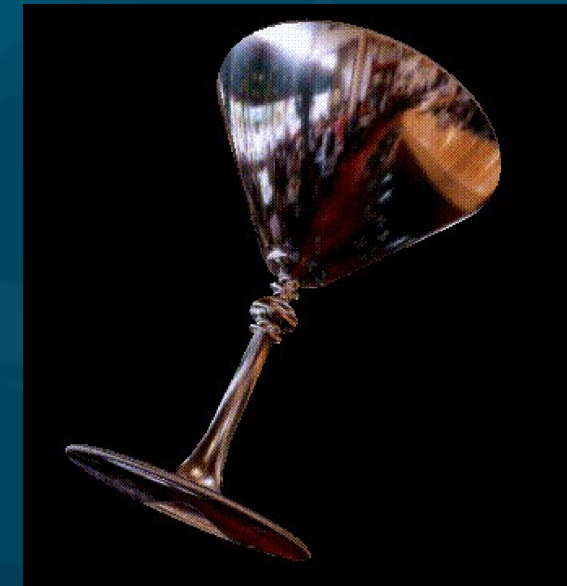
- If mode is GL_EYE_LINEAR, generated texcoords are in the eye (camera) coordinate system
  - Object appears to "swim" in the texture
- Reference plane is specified with
  - glTexGenfv(coord, GL_OBJECT_PLANE, $\{p_1, p_2, p_3, p_4\}$)


Eye coords

Spherical environment maps

- The last mode of auto texcoord generation is GL_SPHERE_MAP:
- Photograph a large silvered ball, or use a fisheye wide-angle lens
- Use automatic spherical texcoords for both s and t:
  - glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP)
- Assumes environment is far away (e.g., small object in large room)





Multitexturing

- Recent OpenGL implementations allow cascading application of multiple textures
- Texture blending function is important
- Specifies how to combine:
  - Previous colour (either from lighting model or from previous textures)s
  - Current texture colour

Compositing and blending

- Learning to use the A (alpha) component of RGBA for:
  - Blending of translucent surfaces
  - Compositing images
    - e.g., layering several textures
  - Antialiasing
    - e.g., motion blur, depth of field

Opacity and transparency

- RGBA: alpha (A) is opacity: 1=opaque, 0=clear
- Say we have an opaque polygon (destination): $(R_d, G_d, B_d, A_d=1)$
- Now blend in a translucent poly (source): (R_s, G_s, B_s, A_s)
- Use source and destination blending factors:
  - $(dR_d+sR_s, dG_d+sG_s, dB_d+sB_s, dA_d+sA_s)$
    - Clamp to [0, 1]
  - One choice: $s=A_s$, $d=(1-A_s)$
  - $((1-A_s)R_d + A_sR_s$, etc...

Blending in OpenGL

- Enable blending in the pipeline:
  - glEnable(GL_BLEND);
- Set source and dest blend factors:
  - glBlendFunc(*src_factor*, *dst_factor*);
- Options for factors: GL_ZERO, GL_ONE, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA, a few others ...
  - glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

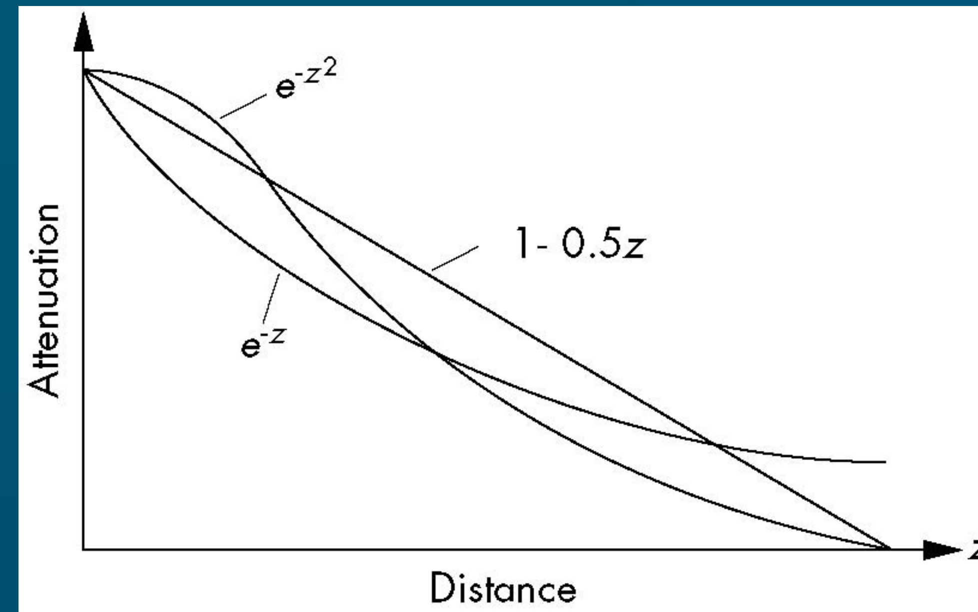
Depth buffer and blending

- By default, polygons are rendered in the order they come down the pipeline (from program)
- Need hidden-surface removal
- Use depth buffer: glEnable(GL_DEPTH_TEST);
  - Tracks depth of each fragment
  - Only renders fragment nearest to camera
- But with blending, this isn't what we want!
- Need to render several fragments
  - Can't use depth buffer for translucent polygons

Mixing translucent and opaque

- Solution: order polygons
- First render all opaque polygons
  - Can use the depth buffer
- Then set the depth buffer to read-only
  - glDepthMask(GL_FALSE);
- Now render translucent polygons in order
  - Must sort them first, back to front

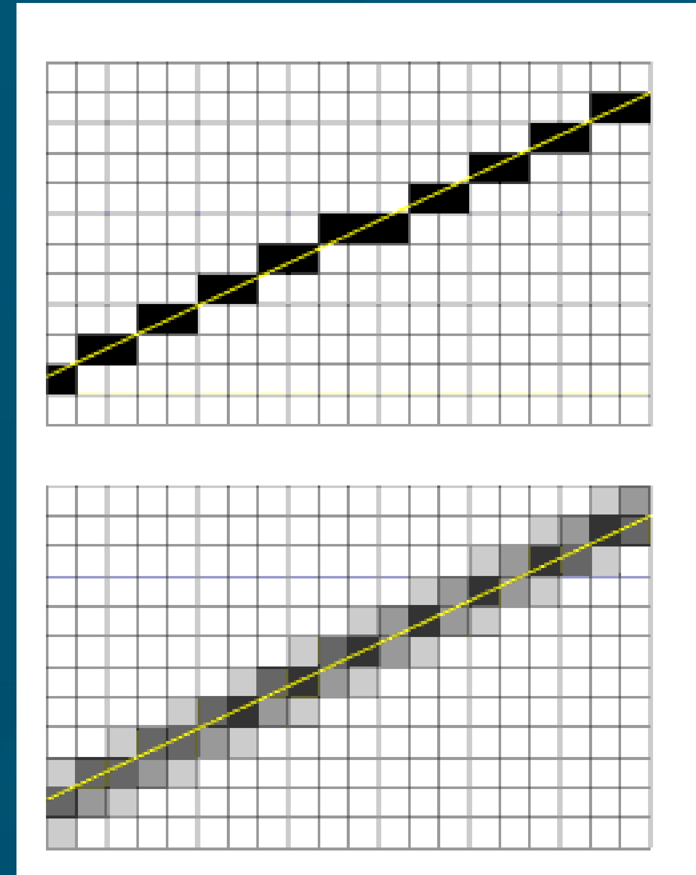
Fog



- Fog is just blending with a fixed fog colour, with a blending function that depends on depth
  - GLfloat fogCol[4] = { 0.5, 0.5, 0.5, 0.5 };
  - glEnable(GL_FOG);
  - glFogf(GL_FOG_MODE, GL_EXP);
  - glFogf(GL_FOG_DENSITY, 0.5);
  - glFogv(GL_FOG, fogCol);
- Fog functions: linear, exponential, Gaussian

Antialiasing

- Aliasing occurs when a mathematical line gets rasterized onto a pixel grid
- With blending enabled, GL can shade a pixel according to how much of the pixel's area is covered by the line
  - glEnable(GL_LINE_SMOOTH);
  - Also for ...POINT..., ...POLYGON...



Accumulation buffer

- Blending involves a lot of math: multiplications and additions
- Limited by the precision of the frame buffer: typically 8 bits per channel (RGBA)
  - Gets worse with number of blends
- Accumulation buffer is a high-precision buffer (usually 16-bit) for blending
  - Read/write into it with a scale factor
  - Copied to framebuffer at the end
  - Slower than compositing directly onto fb

Application: motion blur

- With real-world video cameras, a freeze-frame of a moving scene has motion blur
- Makes motion appear more fluid
- Emulate this in GL:
  - Render object several times as it moves
  - Composite in accumulation buffer

Application: depth of field

- Cameras with aperture are in-focus only in a narrow plane (spherical shell)
- Portions in front of or behind focal point are blurry
- Emulate in GL:
  - Render scene several times with different projection matrices:
  - Same image plane; different centre of projection
  - Composite in accumulation buffer